ICS 321 Fall 2009
SQL: Queries, Constraints, Triggers (ii)

Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa

Find the sid of sailors who have reserved exactly one boat

```
SELECT S1.sid
FROM   Sailors S1
EXCEPT
SELECT R1.sid
FROM   Reserves R1, Boats B1, Reserves R2, Boats B2
WHERE  R1.sid=R2.sid  AND R1.bid=B1.bid
         AND R2.bid=B2.bid  AND R1.bid≠R2.bid
```

```
SELECT R3.sid
FROM   Reserves R3
EXCEPT
SELECT R1.sid
FROM   Reserves R1, Boats B1, Reserves R2, Boats B2
WHERE  R1.sid=R2.sid  AND R1.bid=B1.bid
         AND R2.bid=B2.bid  AND R1.bid≠R2.bid
```

Nested Queries

Q1 : Find the names of sailors who have reserved boat 103

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND bid=103
```

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN ( SELECT R.sid
                  FROM Reserves R
                  WHERE R.bid=103 )
```

- A *nested query* is a query that has another query, called a *subquery,* embedded within it.
- Subqueries can appear in WHERE, FROM, HAVING clauses

Conceptual Evaluation Strategy for Nested Queries

1. Compute the cross-product of *relation-list*.
   - ❑ If there is a subquery, recursively (re-)compute the subquery using this conceptual evaluation strategy
   - ❑ Compute the cross-product over the results of the subquery.
2. Discard resulting tuples if they fail *qualifications*.
   - ❑ If there is a subquery, recursively (re-)compute the subquery using this conceptual evaluation strategy
   - ❑ Evaluate the qualification condition that depends on the subquery
3. Delete attributes that are not in *target-list*.
4. If DISTINCT is specified, eliminate duplicate rows.

Q2: Find the names of sailors who have reserved a red boat

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN ( SELECT R.sid
                  FROM Reserves R
                  WHERE R.bid IN ( SELECT B.bid
                                   FROM Boats B
                                   WHERE B.color=`red’ ))
```

- Unravel the nesting from the innermost subquery

Q21: Find the names of sailors who have not reserved a red boat

```
SELECT S.sname
FROM    Sailors S
WHERE  S.sid NOT IN ( SELECT R.sid
                      FROM Reserves R
                      WHERE R.bid IN ( SELECT B.bid
                                       FROM Boats B
                                       WHERE B.color=`red’ ))
```

Correlated Nested Queries

Q1: Find the names of sailors who've reserved boat #103

```
SELECT S.sname
FROM   Sailors S
WHERE  EXISTS ( SELECT *
                FROM Reserves R
                WHERE R.bid = 103 AND R.sid=S.sid
```

- EXISTS is another set comparison operator, like *IN*.
- If UNIQUE is used, and * is replaced by R.bid, finds sailors with at most one reservation for boat #103. (UNIQUE checks for duplicate tuples; * denotes all attributes. Why do we have to replace * by R.bid?)
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple.

Set Comparison Operators: ANY

- Q22: Find sailors whose rating is better than some sailor called Horatio.

```
SELECT S1.sid
FROM    Sailors S1
WHERE  S1.rating > ANY ( SELECT S2.rating
                         FROM Sailors S2
                         WHERE S2.name=`Horatio’ )
```

- Subquery must return a row that makes the comparison true, in order for S1.rating>ANY to return true

Set Comparison Operators: ALL

- Q23: Find sailors whose rating is better than every sailor.

```
SELECT S1.sid
FROM    Sailors S1
WHERE  S1.rating > ALL ( SELECT S2.rating
                                       FROM Sailors S2
                                       WHERE S2.name=`Horatio’ )
```

- Subquery must return a row that makes the comparison true, in order for S1.rating>ANY to return true

Rewriting INTERSECT Queries using IN

- Q6: Find sid’s of sailors who’ve reserved both a red and a green boat.

```
SELECT S1.sid
FROM    Sailors S1, Boats B1, Reserves R1
WHERE  S1.sid=R1.sid AND R1.bid=B1.bid
      AND B1.color=‘red’
      AND S1.sid IN ( SELECT S2.sid
                      FROM Sailors S2, Boats B2,
                           Reserves R2
                      WHERE S2.sid=R2.sid
                            AND R2.bid=B2.bid
                            AND B2.color=`green’ )
```

Q9: Find the names of sailors who have reserved all boats

```
SELECT S.sname
FROM    Sailors S
WHERE  NOT EXISTS (( SELECT B.bid
                      FROM Boats B )

                     EXCEPT

                    ( SELECT R.bid
                      FROM Reserves R
                      WHERE R.sid=S.sid ))
```

Q9: Find the names of sailors who have reserved all boats (without EXCEPT)

```
SELECT S.sname
FROM    Sailors S
WHERE  NOT EXISTS (( SELECT B.bid
                       FROM Boats B )
                      WHERE NOT EXISTS
                          ( SELECT R.bid
                            FROM Reserves R
                            WHERE R.bid=B.bid
                               AND R.sid=S.sid ))
```

Aggregate Operators

- SQL supports 5 aggregation operators on a column, say A,
  1. COUNT (*), COUNT ([DISTINCT] A)
  2. SUM ([DISTINCT] A)
  3. AVG ([DISTINCT] A)
  4. MAX (A)
  5. MIN (A)

Aggregation Queries

- Q25: Find the average age of all sailors

```
SELECT AVG(S.age)
FROM   Sailors S
```

- Q28: Count the number of sailors

```
SELECT COUNT (*)
FROM   Sailors S
```

- Find the age of the oldest sailor

```
SELECT MAX (S.age)
FROM   Sailors S
```

Q27: Find the name and age of the oldest sailor

```
SELECT S.sname, MAX (S.age)
FROM     Sailors S
```

```
SELECT S.sname, S.age
FROM     Sailors S
WHERE S.age = ( SELECT MAX(S2.age)
                         FROM Sailors S2 )
```

- If there is an aggregation operator in the SELECT clause, then it can only have aggregation operators unless the query has a GROUP BY clause -- first query is illegal.

Queries with GROUP BY and HAVING

```
SELECT      [DISTINCT]  target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
  - The list of attribute names in (i) must be a subset of *grouping-list*.
  - Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group.
  - A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.

Conceptual Evaluation Strategy with GROUP BY and HAVING

- [Same as before] The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, `*unnecessary*' fields are deleted
- The remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* is then applied to eliminate some groups. Expressions in *group-qualification* must have a *single value per group*!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- Aggregations in *target-list* are computed for each group
- One answer tuple is generated per qualifying group

Q32: Find age of the youngest sailor with age >= 18, for each rating with at least 2 such sailors

```
SELECT  S.rating,
           MIN(S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

Conceptual Evaluation for Q32

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

Partition
or
GROUP BY

rating	age
[redacted]	[redacted]
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
[redacted]	[redacted]
[redacted]	[redacted]

Eliminate groups
Using HAVING clause

rating	minage
3	25.5
7	35.0
8	25.5

Perform aggregation
on each group

EVERY and ANY in HAVING clauses

```
SELECT  S.rating, MIN(S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1 AND EVERY ( S.age <=60 )
```

- EVERY: every row in the group must satisfy the attached condition
- ANY: at least one row in the group need to satisfy the condition

Conceptual Evaluation with EVERY

HAVING COUNT (*) > 1 AND EVERY (S.age <=60)

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

Partition
or
GROUP BY

rating	age
[redacted]	[redacted]
[redacted]	[redacted]
7 7	45.0 35.0
8 8	55.5 25.5
[redacted]	[redacted]
[redacted]	[redacted]

Eliminate groups
Using HAVING clause

rating	minage
7	35.0
8	25.5

Perform aggregation
on each group

What is the result of changing EVERY to ANY?

Find age of the youngest sailor with age 18, for each rating with at least 2 sailors between 18 and 60

```
SELECT  S.rating,
            MIN (S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18 AND S.age <= 60
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

Summary

- Nested Queries
  - Correlated nested queries
  - Conceptual evaluation strategy
  - Set comparison operators in WHERE clause: EXISTS, IN, UNIQUE, ANY, ALL
- Aggregation operators: COUNT, MIN, MAX, SUM, AVG
- GROUP BY and HAVING clauses
  - EVERY and ANY in HAVING clause
  - Conceptual evaluation strategy